REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*

2. REPORT TYPE
Journal Article

3. DATES COVERED *(From - To)*

4. TITLE AND SUBTITLE
Time-Parallel Solutions to Ordinary Differential Equations on GPUs with a New Functional Optimization Approach Related to the Sobolev Gradient Method

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Lederman, C; Cambier, J.

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
Q0AE

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory (AFMC)
AFRL/ RQRS
1 Ara Drive
Edwards AFB, CA 93524-7013

8. PERFORMING ORGANIZATION REPORT NO.

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Air Force Research Laboratory (AFMC)
AFRL/RQR
5 Pollux Drive
Edwards AFB, CA 93524-7048

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
AFRL-RZ-ED-JA-2012-186

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES
For publication in Computer Physics Communications (October 2012)
PA Case Number: 12418; Clearance Date: 6/7/2012

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON** J. L. Cambier
a. REPORT **Unclassified**	**b. ABSTRACT** **Unclassified**	**c. THIS PAGE** **Unclassified**	SAR	25	**19b. TELEPHONE NO** *(include area code)* N/A

Time-Parallel Solutions to Ordinary Differential Equations on GPUs with a New Functional Optimization Approach Related to the Sobolev Gradient Method

Carl Lederman and Jean-Luc Cambier
Air Force Research Laboratory, Propulsion Directorate
Spacecraft Branch (AFRL/RZSS)
Edwards AFB, CA

Abstract

The problem of finding a solution to an ODE can by reformulated as a problem of finding the minimum of a specific energy functional. We present an efficient approach to finding this minimum and relate it to the Sobolev gradient method and Newton's method. The proposed approach requires only well studied parallel efficient algorithms in contrast to some other approaches that still require repeated serial solvers but at a lower resolutions. We present examples where, with a very good initial guess of the solution, convergence can be obtained in a single iteration. Even with a very poor guess, only a few iterations are required and convergence is faster than the pure sequential approach. We discussed how the speed of the method decreases rapidly with more parallelization and the method in the context of multi-scale modeling.

1. Introduction

1.A Motivation for the Time-Parallel Approach

The problem is to find a vector $x(t)$ that satisfies the ordinary differential equation (ODE):

$$\frac{dx}{dt} = f(x)$$

$$x(0) = c_0$$

where $f(x)$ is a possibly non-linear function of $x(t)$. The classical approach to solving this problem is to employ a scheme which numerically approximates the ODE [1]. For example, the second order implicit Crank-Nicholson Scheme is given by

$$\frac{x_{m+1} - x_m}{k} = \frac{1}{2} f(x_{m+1}) + \frac{1}{2} f(x_m)$$

$$x_0 = c_0$$

where k is the timestep size and m, an integer, gives the position of x at each discrete time point. Numerically implementing a scheme such as this one leads to perfectly sequential computations. That is, x^4 cannot be computed until x^3 is known, and x^3 is completely unknown until x^2 is computed, etc. This is the most natural way to model time dependent behavior as what occurs in one moment is dependent on what occurred in the previous moment.

But from a modern computational perspective, modeling ODEs in this way is far from ideal. With GPUs (Graphics Processing Units) and multi-core CPUs (Central Processing Units), computations done per second can be massively higher when parallelization can occur. So while a method to model

ODEs that involves parallel computations will almost certainly be less efficient (i.e. requiring more total computations), because some of the computations can be done in parallel, the parallel method could result in greatly reduced computational time. Or put another way, the time parallel approach consist of creating unnatural and less efficient schemes that, nonetheless, are faster because they better utilize existing computational technology. A basic overview of the considerable challenges is given in [2].

1.B Existing Time Parallel Approaches

The Parareal method [3] involves a coarse solution, advanced with a coarse propagator, and fine solution, advanced with a fine propogator. Exactly what the fine and coarse solution are depends on the situation and could include different numerical schemes and even different physical models. But for most of the paper, we focus on the simplest case in which the fine solution is advanced with a smaller time steps size than the coarse solution (which results in the fine solution being more accurate).

The starting point of the method is the coarse solution. This is obtained by sequentially advancing the coarse propagator from the interval $i = 0$ to $i = I$.

$$x_{i+1}^{0} = \mathcal{C}\left(x_i^0\right)$$

After this quantity is computed an iterative procedure is employed, where n is the iteration number. With many common ODEs the number of iterations required is quite small.

On each interval of the time domain, the fine solution and coarse solution are computed across the intervals in parallel and sequentially within each interval. If $\mathcal{F}$ represents the fine propagator of the solution, and $\mathcal{C}$ the coarse propagator, then we have

$${x_f}_{i+1}^{n} = \mathcal{F}(x_i^n) \quad i = 1, \dots, I$$

$${x_c}_{i+1}^{n} = \mathcal{C}(x_i^n) \quad i = 1, \dots, I$$

Note that because the coarse and fine solutions are found independently on each interval, x_f and x_c are not the coarse and fine solutions over the whole time domain. To obtain the fine solution over the whole time domain, the coarse solution is run sequentially over the whole time domain with adjustments made based on the previously computed ${x_f}_{i+1}^{n}$ and ${x_c}_{i+1}^{n}$:

$$x_{i+1}^{n+1} = \mathcal{C}\left(x_i^{n+1}\right) + {x_f}_{i+1}^{n} - {x_c}_{i+1}^{n}$$

The speed of the method depends both on how fast the coarse solution can be computed sequentially, which is performed at each iteration, as well as how quickly the fine solution can be computed in parallel on each interval. This approach is compared with the proposed approach for three example ODEs in the Section 3 where further discussion of the utility of this method is given.

Various modifications to the basic Parareal method have been proposed [4-8], some of which may be particularly well suited to certain types of ODEs. In [8], the authors propose employing the compound wavelet method for adjusting the coarse solution based on the coarse and fine solutions computed in parallel on each interval. This may be particularly useful when the fine solution is noisy. Additionally, the authors claim that in some circumstances this approach may allow the fine time steps to only be computed on a small portion of the time domain. In [6], a method that works well for certain structural dynamics problems is presented and [7] modifies the original Parareal method to better handle cases where large matrices are involved. We note that many of these methods discuss PDEs as well as ODEs. A PDE, when discretized in space, can be written as an ODE with $x(t)$ becoming a very large vector. But there are other issues to consider with PDEs (Partial Differential equations) though, mainly that it is often possible to max out the parallel computing capacity on even a large GPU cluster with only spacial parallelization. The focus of the paper will remain limited to ODEs with some limited discussion of PDEs in the conclusion, Section 4.

Other types of parallel approaches beyond the Parareal method exist as well. The Waveform Relaxation Method [9,10] solves large systems of differential equations by breaking down the system into loosely coupled subsystems. In [2] an approach to solving linear ODEs in parallel is discussed with a means to extend this approach to a few limited types of ODEs.

1.C The functional optimization approach

The problem of finding a solution to an ODE can by reformulated as a problem of finding the minimum the following functional:

$$F\big(\vec{x}(t)\big) = \frac{1}{2}\int_0^T \left(\frac{dx}{dt} - f(x)\right) \cdot \left(\frac{dx}{dt} - f(x)\right) dt$$

When the function *x* that minimizes *F* is found, i.e. *F*=0, then we have:

$$\frac{dx}{dt} - f(x) = 0$$

At the most basic level, the reason for incorporating this functional is that it quantifies how far away any given function is from satisfying the ODE of interest. This quantifying of the error can then be used to determine how to modify the function to make it more closely resemble the true solution of the ODE.

Note that the functional is not convex for non-linear f. Thus an optimization approach that relies too heavily on the convexity assumption of the functional would not be appropriate.

Also of note is that there are, of course, other possibilities for functionals that measure the error in the ODE. But this one is the simplest to work with in many ways.

1.D The L^2 Gradient

The L^2 gradient of the functional is:

$$\nabla_{L^2} F(x) = -df|_x{}^T \left(\frac{dx}{dt} - f(x)\right) - \frac{d}{dt}\left(\frac{dx}{dt} - f(x)\right)$$

This can be computed directly from the functional [11]. Note that here and in the rest of the paper we do not specify boundary conditions. The only boundary condition is at the point $x(0)$ and it remains fixed.

One approach to finding the functional minimum is to solve for when this derivative is 0 (the Euler-Lagrange equation) [11].

$$\nabla_{L^2} F(x) = 0$$

But this involves solving a large non-linear system where the size is based on the time step number. As an alternative, a popular approach is to employ gradient descent [11]:

$$\frac{\partial x(t,v)}{\partial u} = -\nabla_{L^2} F(x(t,v))$$

If implemented explicitly, no linear system needs to be solved.

$$\frac{x_m^{n+1} - x_m^n}{l} = -\nabla_{L^2} F(x(t,v))$$

The artificial time step size is given by l and each vector x now has two coordinates, the time coordinate m and the artificial time coordinate n.

But this approach yields at least two major problems that make it impractical. The first issue is the severe size limitation on the artificial time step size. The L^2 gradient of F has a second derivative in time and thus the artificial time step size would need to scale as the square of the time step size. The other issue is the speed of propagation of information. If the values of x change near time equal 0, the values near the end of the time domain will not be affected by that change at all until many artificial time steps have occurred.

1.E The Sobolev gradient

The Sobolev Gradient can replace the L^2 gradient and leads to a faster numerical computation of a functional minimum [12-20]. The theory behind this approach is well established [12]. One way this has been explained is by comparing the standard L^2 inner product and the most common Sobolev inner product, the H^1 inner product [12,13]. The L^2 inner product is given by:

$$\langle y, z \rangle_{L^2} = \int_0^T y(t) \cdot z(t) dt$$

This inner product is assumed when computing the L^2 gradient of a functional. The H^1 inner product is given by:

$$\langle y, z \rangle_{H^1} = \int_0^T y(t) \cdot z(t) dt + \int_0^T \frac{dy(t)}{dt} \cdot \frac{dz(t)}{dt} dt$$

In general, the Sobolev inner product depends only on the two functions and their derivatives. Higher order derivatives can be appropriate if the functional of interest contains higher order derivatives[13].

The H^1 gradient is not computed directly from a functional. Instead, the L^2 gradient is computed and then modified based on the relationship between the L^2 inner product and H^1 inner product. With the integration by parts formula:

$$\langle y,z\rangle_{H^1} = \int_0^T \left(I - \frac{d^2}{dt^2}\right) y(t)\cdot z(t)dt$$

Which defines a relationship between the H^1 and L^2 inner products.

$$\langle y,z\rangle_{H^1} = \langle\left(I - \frac{d^2}{dt^2}\right) y,z\rangle_{L^2}$$

Taking the directional derivative (Fréchet derivative) in the different norms gives [12,13] :

$$F^{'}(x)h = \langle\nabla_{L^2}F(x),h\rangle_{L^2}$$

$$F^{'}(x)h = \langle\nabla_{H^1}F(x),h\rangle_{H^1}$$

$$\langle\nabla_{L^2}F(x),h\rangle_{L^2} = \langle\nabla_{H^1}F(x),h\rangle_{H^1} = \langle\left(I - \frac{d^2}{dt^2}\right)\nabla_{H^1}F(x),h\rangle_{L^2}$$

$$\nabla_{H^1}F(x) = \left(I - \frac{d^2}{dt^2}\right)^{-1}\nabla_{L^2}F(x)$$

Thus, to obtain the H^1 gradient, and the Sobolev gradient in general, the L^2 gradient is computed and then a sparse linear system must be solved for. The H^1 gradient is smoother than the L^2 gradient and allows much larger time steps to be taken. This stems from the fact the Sobolev inner product depends on the derivatives of the input functions while the L^2 inner product does not. This approach has been shown effective in a wide range of applications, including physical modeling [14], image segmentation and registration [15-17], and tetrahedral mesh generation [18].

The Sobolev gradient has been examined specifically in the context of ODEs [19,20]. While shown to be far better than the L^2 gradient, a few major problems still exist. One issue is that it takes too many iterations to converge. The number of iterations is large enough that it is hard to argue that it is faster than the sequential approach under any reasonable computing system. Additionally, the method does not immediately parallelize well. This is due to the tridiagonal system that must be solved. While these systems can be solved quickly with CPUs [21], when solving in parallel a less efficient iterative scheme must be employed [22]. Some of these problems were considered in [20] where it was proposed that the time domain could be broken down into subintervals and variable time step sizes employed. But many of the general strategies presented could apply to other optimization approaches and further work is still required to make an approach like this competitive in computation time with the sequential approach.

What is needed to make an optimization approach like the Sobolev gradient method viable is an orders of magnitude decrease in the iteration count as well as a construction that relies only on algorithms that can be done efficiently and directly with parallel computing. We describe such an approach in the methods section below.

2. Methods

2.A Derivation base on the Sobolev Gradient Method

While the Sobolev gradient produces a smoother descent that the L^2 gradient, it still does not prevent x from updating to configurations that differ significantly from the true solution of the ODE. So, as a starting point, we consider an inner product that matches the functional $F(x)$ as closely as possible:

$$\langle y,z\rangle_{\widetilde{ODE}} = \int_0^T \left(\frac{dy}{dt} - f(y)\right)\cdot\left(\frac{dz}{dt} - f(z)\right)dt$$

This inner product is not, by definition, a Sobolev inner product. A Sobolev inner product depends only on the input functions and their derivatives where as this inner product depends on f. Nonetheless, the methodology of the Sobolev gradient can still be applied to produce an effective numerical approach to solving ODEs. But first, the functional needs to be modified slightly in order to remove the non-linearity present in f. We note that we are only considering the first derivate of $F(x)$. Therefore, nothing is really gained by considering higher order changes in the constituent parts of the functional. Also, everything that is computed when obtaining the gradient is based on a specific configuration of x at a fixed artificial time. Thus, we consider a modified inner product that only depends on the linear operator df which is evaluated based on the current value of x.

$$\langle y,z\rangle_{ODE}(x) = \int_0^T \left(\frac{dy}{dt} - df|_x y\right)\cdot\left(\frac{dz}{dt} - df|_x z\right)dt$$

From this inner product, the way to alter the L^2 gradient can be derived by following the same approach as is done with the Sobolev Gradient:

$$\langle y,z\rangle_{ODE}(x) = \int_0^T \frac{dy}{dt}\cdot\left(\frac{dz}{dt} - df|_x z\right) - df|_x y\cdot\left(\frac{dz}{dt} - df|_x z\right)dt$$

$$\langle y,z\rangle_{ODE}(x) = \int_0^T y\cdot\left(-\frac{d^2z}{dt^2} + \frac{d}{dt}df|_x z - df|_x{}^T\frac{dz}{dt} + df|_x{}^T df|_x z\right)dt$$

$$\nabla_{ODE}F(x) = \left(-\frac{d^2}{dt^2} + \frac{d}{dt}df|_x - df|_x{}^T\frac{d}{dt} + df|_x{}^T df|_x\right)^{-1}\nabla_{L^2}F(x)$$

This expression is referred to as the ODE gradient simply because the solution to the ODE was the motivation for the selection of the inner product that leads to this gradient. It can be simplified considerably by factoring the operators as follows:

$$= \left(-\frac{d}{dt} + df|_x\right)^{-1}\left(\frac{d}{dt} + df|_x{}^T\right)^{-1}\left(-\frac{d}{dt} - df|_x{}^T\right)\left(\frac{dx}{dt} - f(x)\right)$$

$$= \left(\frac{d}{dt} - df|_x\right)^{-1}\left(\frac{dx}{dt} - f(x)\right)$$

An efficient numerical scheme to compute this quantity is given is sections 2.C and 2.D, but first an alternative derivation is given in the section below.

2.B Derivation based on Newton's Method

The same formula can be obtained by finding the x that satisfies

$$G(x) = \frac{dx}{dt} - f(x) = 0$$

using Newton's method. In order to accomplish this, G is discretized. For example, with Crank-Nicholson, we have

$$G_m = \frac{x_{m+1} - x_m}{k} - \left(\frac{1}{2} f(x_{m+1}) + \frac{1}{2} f(x_m)\right) \qquad m = 0, \dots, M$$

Newton's method is given by:

$$x_{next} = x - dG(x)^{-1} G(x)$$

Where dG is the Jacobian of G. This formula could be used to obtain the solution of an ODE. But solving large block tridiagonal systems using parallel computing is not efficient, as discussed. A better approach is given in the sections below, but first the contents of the Jacobian need to be interpreted and converted back to the continuous setting:

$$x_{next} = x - \left(\frac{d}{dt} - df|_x\right)^{-1} \left(\frac{dx}{dt} - f(x)\right)$$

This derivation is included because it is based on Newton's method, which is much more widely known than the Sobolev gradient method, and gives some indication of why the convergence of the method is so fast. But, while we are merely generous with our modifications of the Sobolev Gradient method in Section 2.A, no functional analysis arguments are present here. We want $G(x)$ to be close to 0, and have declared how it should be updated based on other functions and operators on functions, but no measure of the size of $G(x)$ is made. As the discussion of the Sobolev gradient method should make clear, the norm employed to measure the size of a function can have a significant effect. We note that the functional version of Newton's method can be employed on the functional given in Section 1.C, but this results in a more complicated expression.

2.C. Analytic Expression for ODE Gradient

The formula for the ODE Gradient

$$\nabla_{ODE} F(x) = \left(\frac{d}{dt} - df|_x\right)^{-1} \left(\frac{dx}{dt} - f(x)\right)$$

can be rearranged as:

$$\left(\frac{d}{dt} - df|_x\right) \nabla_{ODE} F(x) = \left(\frac{dx}{dt} - f(x)\right)$$

This linear ODE has a known analytical solution, namely:

$$\nabla_{ODE} F(x) = \int_0^t \exp\left(\int_u^t df|_x(r)dr\right)\left(\frac{dx}{du} - f(x(u))\right)du$$

At first glance, this appears to not be a particularly useful expression. It involves a matrix exponential and computing matrix exponentials is a challenging and still current area of research [23]. But any standard sequential numerical scheme defines precisely how to compute a matrix exponential. The solution to the linear ODE:

$$\frac{dx}{dt} = M(t)x(t)$$

is given by:

$$x(t) = \exp\left(\int_0^t M(r)dr\right)$$

If this same linear ODE were to be solved for with a scheme such as Crank-Nicholson, we have:

$$x^M = \prod_{m=0}^{M} O_m x^0 = \prod_{m=0}^{M} \left(I - \frac{k}{2} M_{m+1}\right)^{-1} \left(I + \frac{k}{2} M_m\right) x^0$$

This makes it clear that the best way to compute the exponential of an integral numerically is to multiply the linear operators which depend on the numerical scheme being employed. The next section describes an efficient way to add the vectors and multiply the operators using parallel computing resources.

2.D Parallel Efficient Approach to Computing the ODE Gradient

The Scan procedure is defined by [24,25]:

$$[a_1, a_2, \dots, a_N] \quad \xrightarrow{Scan \oplus} \quad [\widetilde{a_1}, \widetilde{a_2}, \dots, \widetilde{a_M}] = [a_1, (a_1 \oplus a_2), \dots, (a_1 \oplus a_2 \oplus \dots \oplus a_M)]$$

While trivial when done in serial, a parallel version contains some subtleties. Fortunately, the approach has been carefully developed [24,25] and other issues not considered here such as how to handle non power of two arrays has been researched. In summary, implementing the Scan procedure involves constructing two tree structures in order to perform many operations in parallel. A pictorial example is shown in Figure 1. Scan requires $O(M)$ total computations, but only $O(\log_2 M)$ sequential operations.

Performing Scan in parallel becomes faster relative to the serial version as the size of M increases. This partly to due to the fact that the computational cost of the tree infrastructure is significant relative to the cost of computing the operations, $\oplus$, for small M and because $O(\log_2 M)$ is significant relative to the size of $O(M)$ for small M which limits the amount of parallelization that can occur. In practice, we obtained reasonably fast results with our Scan procedure with only a few thousand time steps and greater improvements as M increased. The Scan operations for the proposed method are matrix multiplication, which is performed twice at each iteration for both the operators and inverse operators, and vector addition.

2.E Summary of Method

The method can be summarized in the following two lines:

$$\nabla_{ODE}F(x(t)) = \int_0^t \exp\left(\int_u^t df|_x(r)dr\right)\left(\frac{dx}{du} - f(x(u))\right)du$$

$$x_{next} = x - \nabla_{ODE}F(x(t))$$

The discrete approximations of all continuous quantities are based on the sequential scheme of the ODE. For example, with the Crank-Nicholson scheme, we have the values directly apparent in the scheme:

$$\frac{da(km)}{dt} = \frac{a_{m+1} - a_m}{k}$$

$$f(a(km)) = \frac{1}{2}f(a_{m+1}) + \frac{1}{2}f(a_m)$$

Other values are determined based on the scheme. An integral is the inverse of the derivative:

$$\int_0^M (a(t))dt = \sum_{n=0}^{n=N} a_n$$

And the exponential of the integral of matrices, as discussed, is give by:

$$\exp\left(\int_0^N M dt\right) = \prod_{n=0}^{N}\left(I - \frac{k}{2}M_{n+1}\right)^{-1}\left(I + \frac{k}{2}M_n\right)$$

The matrix products and vector sums are computed efficiently with parallel computing resources using the Scan procedure discussed in section 2.D. This makes most of the computations purely parallel, that is $O(M)$ total operations and $O(1)$ sequential operations, and the rest are $O(M)$ total operations and $O(\log_2 M)$ total operations.

3. Results and Discussion

3.A Setup for comparing methods

Three different non-serial methods are compared for solving ODEs, the proposed ODE Gradient method, the Parareal method [3], and the Sobolev Gradient with Scan method. The Sobolev Gradient with Scan method relies on the Scan procedure described in Section 2.D rather than obtaining the solution to a tridiagonal system. This modified approach converges in a comparable number of iterations as the standard implementation of the Sobolev Gradient, but is faster on the GPU. Repeatedly solving the tridiangonal system that emerges from the Sobolev Gradient method can be done simply and efficiently on a CPU with a highly sequential algorithm [21], but requires less efficient approaches on the GPU [22]. Since this iterative tridiagonal solver can be avoided, we do so, to the make the fairest comparison with the proposed method.

To evaluate these methods, we consider a standard approach that involves a coarse solution, which can be computed relatively quickly sequentially and is qualitatively correct, and a fine solution, which is more accurate but takes considerably longer to compute sequentially [3,8]. The coarse solution is given as an input to each of the three 3 non-serial methods, which then should return a very close approximation to the fine solution. For the two optimization based methods, Sobolev Gradient with Scan and the proposed ODE Gradient, the coarse solution is simply entered as an initial guess where the empty values are filled in using linear interpolation. For the Parareal method, the coarse solution is updated repeatedly by running the fine propogators in parallel as discussed in Section 1.B.

The fine step number was fixed at 32768. The coarse step numbers varied from 4 to 16384. This spans a large range of approximations, to the quick and poor, to the slow but fairly accurate. The sequential solution with 32768 steps was considered the silver standard, and the non-serial methods were iterated until the error relative to the silver standard at the last time point was less than one tenth the error from considering 16384 steps in serial. This balance was chosen to give some justification to employing the full 32768 time steps but at the same time avoid considering the region of potentially slow convergence that comes about due to double precision floating point operation error in all three of the methods. Figures 4, 8, and 11 give the number of iterations each method requires to converge under this criterion.

The computation time of all three methods was measured. While the timing depends on the computing system and the encoding, the orders of magnitude differences seen amongst the methods does give some insight on their effectiveness. Computations for the Sobolev gradient with Scan and ODE gradient methods were performed on a single graphics processor card. Both of these methods rely only on parallel computing algorithms. The Parareal method depends on both parallel computations and a sequential operation and so was allowed to employ a single core of a CPU in addition to the GPU. The speed increase when performing purely parallel computations on the GPU was 45 times faster than on the CPU. Since solving an ODE is not even close to a purely parallel computation, the speed up should be much less than this. Figures 5, 9, and 12 report the speed increase of each of the non-serial methods compared with the sequential approach on the CPU (that is, computation of time of serial divided by computation time of non-serial).

The computation time is averaged over ten runs for each situation and restricted to the pure computation time of each method. That is, code compiling, allocating $O(M)$ size arrays, running the coarse solution in serial that is given as input to each method, and plotting issues with CPUs and GPUs are not included in the time. These are all important areas of research that may affect the computation time in practice, but are outside the scope of the current research.

3.B Example ODEs

Three well known ODEs are considered. The first is the Lotka-Volterra equation with a quadratic decay term, given by:

$$\frac{dx_1}{dt} = 10x_1 - .01x_1x_2 - .001x_1x_1$$
$$\frac{dx_2}{dt} = .01x_1x_2 - 10x_2 - .0002x_2x_2$$

$$x_1(0) = 500, x_2(0) = 500, t \in [0,5]$$

This model represents the population of two interacting species, a predator and prey. The quadratic decay term causes decay to fixed population sizes and avoids exact repetition of values over several cycles. The scheme is implemented using Crank-Nicholson. Figures 2 and 3 show phase space plots of the most accurate approximation to the solution (32768 steps) in black and approximations in cyan and magenta.

The second ODE is the pendulum equation, given by:

$$\frac{dx_1}{dt} = x_2$$

$$\frac{dx_2}{dt} = -\sin(x_1)$$

$$x_1(0) = .5, x_1(0) = 2, t \in [0,20]$$

This ODE was also implemented using Crank-Nicholson. Phase space plots are given in Figures 6 and 7.

The last ODE is the non-linear and stiff Van der Pol equation:

$$\frac{dx_1}{dt} = x_2$$

$$\frac{dx_2}{dt} = -100(x_1x_1 - 1)x_2 - x_1$$

$$x_1(0) = 2.0, x_1(0) = 0, t \in [0,1]$$

A phase-space plot is given in Figure 10. Because the ODE is stiff, the scheme employed was Backward Euler. The Sobolev gradient method was not included in this example. While the Sobolev Gradient method has been shown to work on stiff ODEs (and this ODE in particular) [20], it requires carefully breaking up the domain into intervals and varying the timestep size, which is beyond the scope of the straightforward implementations done here.

For this example, the product of the linear operators becomes invertible to machine precision. This is handled in a relatively straightforward way for the proposed method with the following two modifications. First, the functional energy is monitored and when there is convergence to machine precision up to a certain time, the operators are replaced with the identity. This has the effect of starting to solve the ODE at a later time at higher iteration numbers. Second, the pseudo-inverse of the inverse operators is taken rather than the true inverse. This has the effect of zeroing out certain values of the sum and producing a slightly worse approximation of the integral computed for the ODE gradient. As is shown in Figures 11 and 12, these modifications had only a small effect on performance.

3.C Discussion

In general, all of the methods converged within a reasonable amount of time. One area of exception to this, for all methods, was when the coarse time steps were very large. But we do not view this as significant because using large time steps is perilous, even when considering the direct serial approach. Even implicit methods, which in theory have no timestep size restrictions, may have problems in practice with Newton’s method converging at each step unless an elaborate and computationally more expensive implementation is employed.

In general, the Sobolev Gradient method with Scan took more iterations and more computation time to converge than the proposed ODE Gradient method. When compared to the sequential approach, the Sobolev Gradient with Scan method took more time while the ODE Gradient method took less time. Also of note is that even when the Sobolev Gradient method was given a solution very close to the true solution (i.e. initialized with the sequential method using 16384 time steps and trying to approximate the solution with 32768 time steps), it still took many iterations to converge. This has to do with norm associated with the gradient. The update of the solution at a given iteration using the Sobolev Gradient may be the appropriate amount to update in the Sobolev norm but still results in a significant increase in error in the ODE. For the Lotka-Volterra equations in particular, $x(t)$ became infinitely valued near the end of the time domain before convergence. The time domain was broken up into 8 intervals, as suggested in [20], which adequately dealt with the problem.

The number of iterations required for the Parareal method and the ODE gradient was generally very similar despite the fact that they are very dissimilar mathematically. A close approximation to the high resolution solution resulted in few iterations needed, and the number of iterations increased only slowly as the accuracy of the coarse solution deteriorated. The computation time required for the ODE gradient is simply roughly proportional to the number of iterations. The interpretation of the computation time required for the Parareal method is a bit more complicated. The speed is slow when a close approximation to the fine solution is given. In this case, the serial method must be run over a large number of time steps. The speed is also slow when a very coarse solution is given. In this case the fine solution, when run on each interval requires many serial steps. The computation time of Parareal method is comparable to the ODE gradient method in a specific range where the coarse step to fine step ratio is in the right balance. This balance depends upon the speed of sequential computation, the number of parallel processors, and the speed of each parallel processor. In some applications determining and implementing a good choice for coarse and fine solutions may not be a significant issue, and the desired speed increase may be achieved. We discuss in Section 4.B some general types of applications where the ODE gradient method may be a better choice.

4. Conclusion

4.A Summary of Comparison with Sequential Approach

We have described an efficient means to compute ODEs on GPUs. The method compares well with two other existing non-serial approaches, the Parareal Method and the Sobolev Gradient method. The speed increase over the sequential approach is a result of the method being able to take advantage of parallel computing where more total computations per millisecond can be performed compared to serial computing. One of the premises discussed in the introduction was that the sequential approach is always more efficient, assuming no speed up from parallel computing. We explain below exactly how the proposed method is less efficient.

The simplest explanation is that the proposed approach is an iterative approach and the sequential approach is not. But this is not entirely correct, at least for implicit schemes. The sequential approach requires Newton iterations at each time step where as the proposed approach does not. It simply does not appear in the formulation. Many small iterations at each time step are replaced by a few iterations on the scale of the entire function.

The drawback of approaches like the one proposed can be observed with a very simple example. Suppose vector x_4 is found by applying 4 linear operators to x_0:

$$x_4 = O_4 O_3 O_2 O_1 x_0$$

The value of x_4 can be obtained with the following four sequential operations:

$$x_0 \to O_1 x_0 = x_1 \to O_2 x_1 = x_2 \to O_3 x_2 = x_3 \to O_4 x_3 = x_4$$

Alternatively, x_4 can be found with 4 operations but only three sequential operations.

$$\begin{matrix} O_4 O_3 \to O_{43} \\ O_2 O_1 \to O_{21} \end{matrix}, \quad x_0 \to O_{21} x_0 = x_2 \to O_{43} x_2 = x_4$$

But multiplying matrices together is more computationally costly than matrix vector multiplication. If the dimension size is d, matrix multiplication is $O(d^3)$ operations where as matrix vector multiplication is $O(d^2)$. This would suggest the proposed method is a better choice only when the time step number, M, is much larger than the dimension size, as the number of sequential operations scales as $O(\log(M))$ at the cost of $O(d)$.

There are important cases where the size of the system of ODEs is large, most noteably, the spatial coordinates of a PDE can be discretized to create a very large dimensional system of ODEs. The proposed method, as it currently stands, would not be a good choice as it would require multiplying these large matrices together. However, it is not immediately clear that any time parallel method would be a suitable choice. PDEs can be parallelized in their spatial coordinates in a much more straightforward way. For example, large matrix vector multiplication can be trivially parallelized. Data transfer speeds, whether the computation is done on GPUs or multi-core CPUs, etc. all need to be considered. So while we certainly would not rule out any application of the proposed approach to PDEs, we see ODEs as the most direct application.

4.B Potential Applications

A diverse selection of approaches exists for solving ODEs and the appropriateness of the approach depends heavily on the specific type of application. Some of the applications where the proposed approach may be appropriate are discussed below.

A coarse and fine representation of a model may exist that depend upon specific numerical schemes or physical models. In this case, the coarse and fine scales cannot be chosen arbitrarily to fit the existing computational resources. The proposed method is relatively insensitive to the scales of the coarse and fine solution and so this may be an appropriate approach.

A machine may need to take inputs, model an ODE in real time, and respond. Since a real time response is required, speed is paramount, and the proposed method is fast. Furthermore, the proposed method can quickly make small increases in accuracy (by doubling the number of time steps and iterating once, for example) which may be useful if a machine has a solution but only a small amount of time to increase accuracy as much as possible before a response is required.

Lastly, some experiments may require running a large number of ODEs. For example, a chemist might have some rate equations that depend on various parameters or models which need to be altered in complex ways depending on the results of previous runs.

The proposed approach modifies the Sobolev gradient method to solve ODEs in less time. It compares well to the existing Parareal method. We have carefully described what types of ODEs it would be best suited to solve as well as given several potential general applications.

References

[1] Iserles Arieh, "A First Course in the Numerical Analysis of Differential Equations" Cambridge University Press, 2nd edition. (2008)

[2] Gear, C.W., Parllel Methods for Ordinary Differential Equations, Vector and Parallel Processors for Scientifici Computing -2, Accademia Nazionale dei Lincei and IBM, Rome, September 1987.

[3] J.-L. Lions, Y. Maday, G. Turinici, A parareal in time discretization of pde's, Comptes Rendus de l Academie des Sciences, Paris, Serie I 332 (2001) 661–668.

[4] D. Samaddar, D.E. Newman, R. Sánchez, "Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm," J. Comput. Phys. 229 (18) (2010) 6558.

[5] Harden, C. and Peterson, J. "Combining the parareal algorithm and reduced order modeling for time dependent partial differential equations" Author on-line post.

[6] Farhat, C. and Cortial, J. "A Time-Parallel Implicit Method for Accelerating the Solution of Nonlinear Structural Dynamics Problems" International Journal for Numerical Methods in Engineering. 0:1-25 (2008).

[7] P.Amodio and L.Brugnano. Parallel solution in time of ODEs: some achievements and perspectives. Appl. Numer. Math. 59 (2009) 424–435.

[8] G. Frantziskonis, K. Muralidharan, P. Deymier, S. Simunovic, P. Nukala, and S. Pannala. Time-parallel multiscale/multiphysics framework. Journal of Computational Physics, 228(21):8085 – 8092, 2009.

[9] G, Martin. "AWaveformRelaxation Algorithm with Overlapping Splitting for Reaction Diffusion Equations: Numerical Linear Algebra with Applications, Vol. 1(1), 1-7 (1993)

[10] J. Sand and K. Burrage. "A Jacobi Waveform Relaxation Method for ODEs". SIAM Journal on Scientific Computing, 20(2):534– 552, 1998. ISSN 1064-8275.

[11] Fox, C. An introduction to the calculus of variations. Courier Dover Publications. (1987).

[12] Neuberger, J.W. "Sobolev Gradients and Differential Equations". Lecture Notes in Mathematics #1670. Springer. (1997)

[13] R. J. Renka, Constructing fair curves and surfaces with a Sobolev gradient method, CAGD 21 (2004), 137-149.

[14] Majid, A., and Sial, S. "Approximate solutions to Poisson–Boltzmann systems with Sobolev gradients", Journal of Computational Physic Vol 230, Issue 14, 5732-5738 (2011).

[15] Renka, R.J. "Image Segmentation with a Sobolev Gradient Method" Nonlinear Analysis 71, 774-780. (2009)

[16] Lin, T., Dinov, I., Toga, A., Vese., L. "Nonlinear Elasticity Registration and Sobolev Gradients" Biomedical Image Registration Vol 6204 269-280 (2010).

[17] Lederman, C., Vese, L., and Chien, A., "Registration for 3D Morphological Comparison of Brain Aneurysm Growth" Advances in Visual Computing Vol 6938 392-399 (2011).

[18] Lederman, C., Joshi, A., Dinov, I., Vese, L., Toga., A., and Van Horn, J.D. "The Generation of Tetrahedral Mesh Models for Neuroanatomical MRI". Vol 55 Issue 1. 153-164 (2011).

[19] Mahavier, W. T., "Solving boundary value problems numerically using steepest descent in Sobolev spaces." Missouri J. of Math. Sci. 11(1):19-32.

[20] Mujeeb, D., Neuberger, J.W., and Sial, S. "Recursive Form of Sobolev Gradient Method for ODEs on Long Intervals". International Journal of Computer Mathematics. Vol 85, Issue 11, (2008)

[21] Conte, S.D., and deBoor, C. (1972). Elementary Numerical Analysis. McGraw-Hill, New York

[22] Parallel Tridiagonal Equation Solvers, Harold S. Stone ACM Transactions on Mathematical Software (TOMS), Volume 1 Issue 4, Dec. 1975

[23] Higham, N. "The scaling and squaring method for the matrix exponential Revisited". In SIAM Journal of Matrix Analysis Applications, no. 4, 1179–1193. (2005)

[24] Harris, Mark. "Parallel Prefix Sum (Scan) with CUDA" NVIDIA GPU Computing Documentation. April 2007

[25] Guy E. Blelloch. "Prefix Sums and Their Applications". In John H. Reif (Ed.), Synthesis of Parallel Algorithms, Morgan Kaufmann, 1990.

Figures

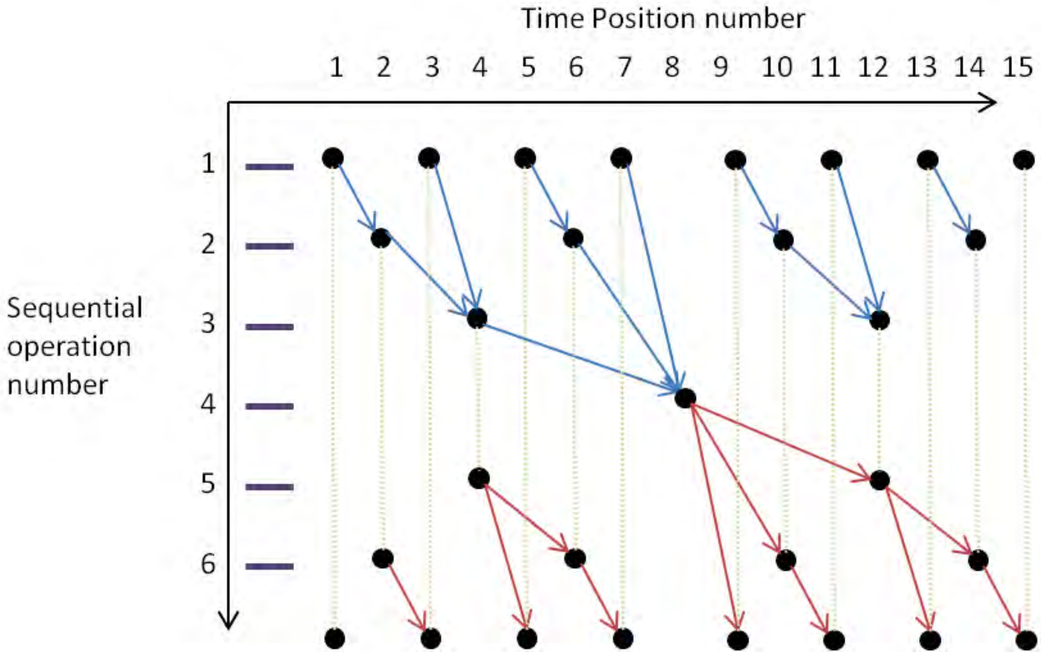

Figure 1. A rough pictorial summary of the Scan procedure. The dots are discrete values on the time domain and the arrows represent an operation. The purple line indicates a set of operations that are done sequentially.

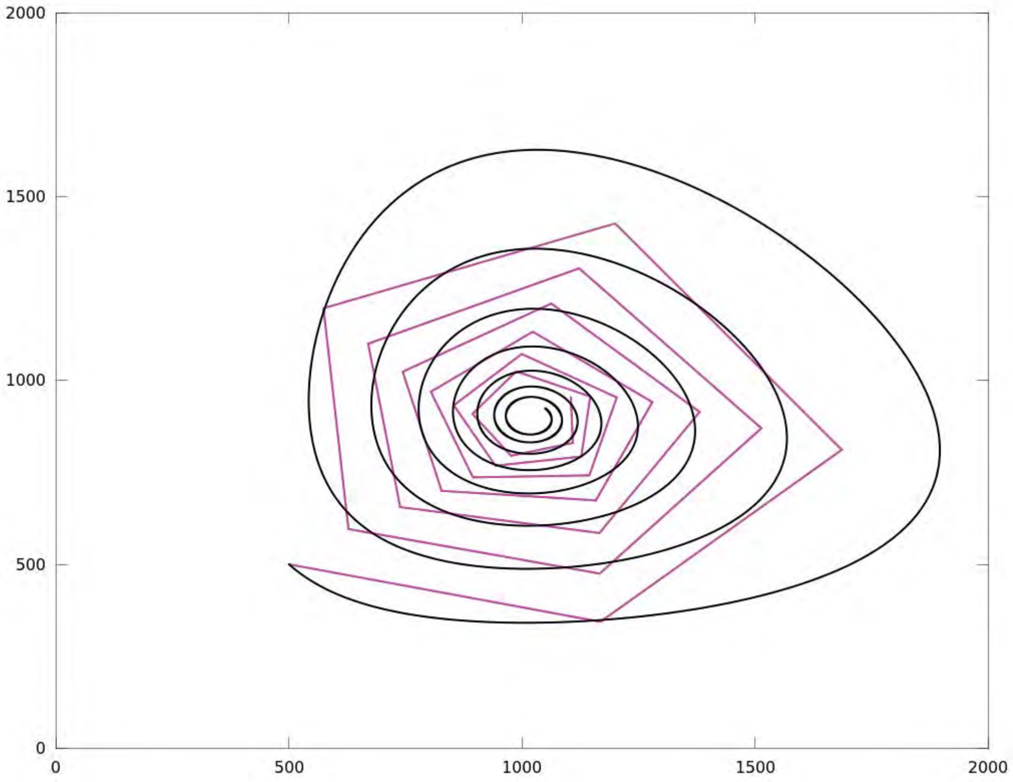

Figure 2. A phase space plot of the Lotka-Volterra example. Fine solution (black) contains 32768 time steps while the coarse solution (magenta) contains 32 time steps. The non-serial methods that converged are all visually indistinguishable from the fine solution computed in serial.

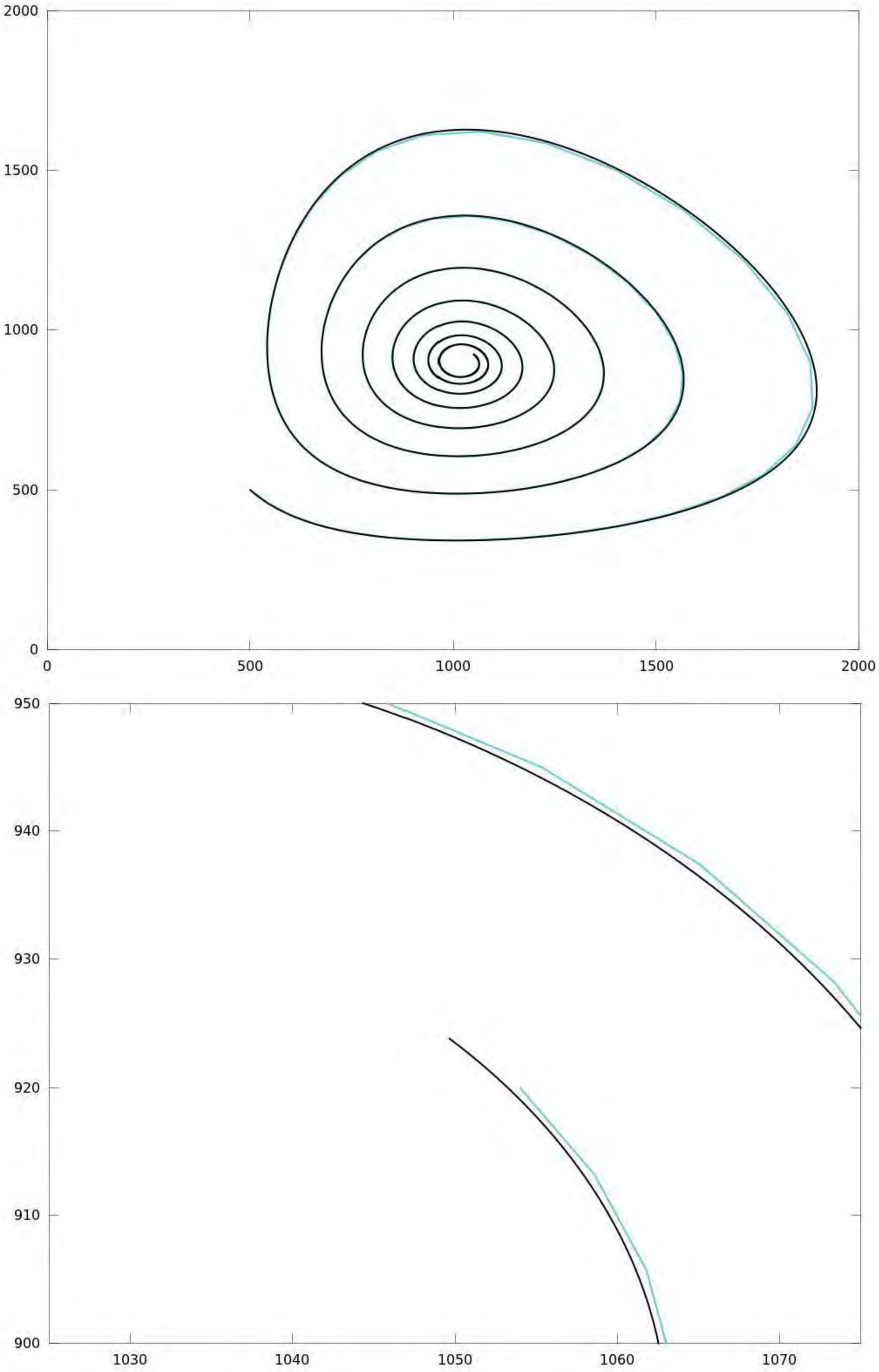

Figure 3. A phase space plot of the Lotka-Volterra example. Fine solution (black) contains 32768 time steps while the coarse solution (cyan) contains 256 time steps. The non-serial methods that converged are all visually indistinguishable from the fine solution computed in serial.

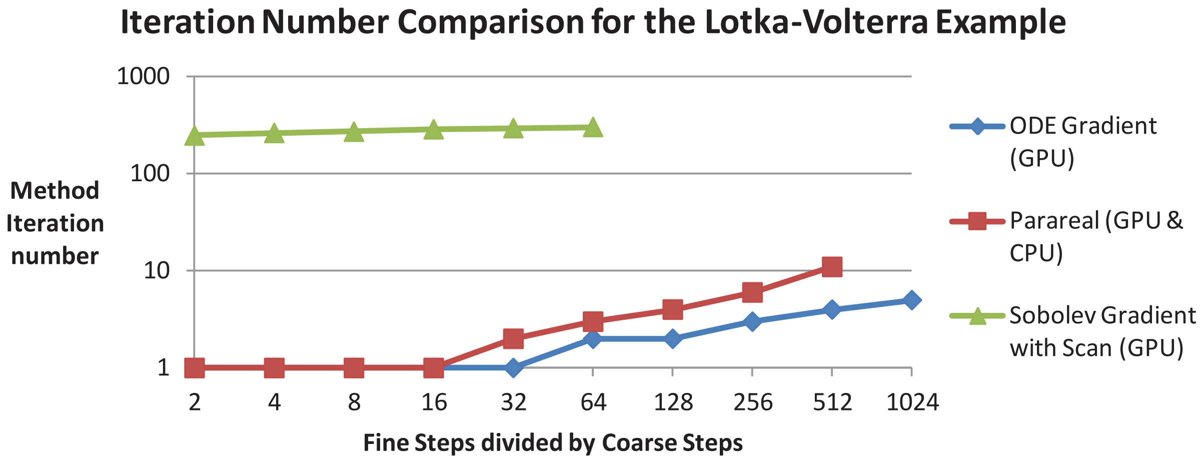

Figure 4. The number of iterations required for each of the non-serial methods. The fine time step number is fixed at 32768 while the coarse step number is varied. The iterations are stopped when the error at the last time point is less than one tenth the error with 16384 time steps.

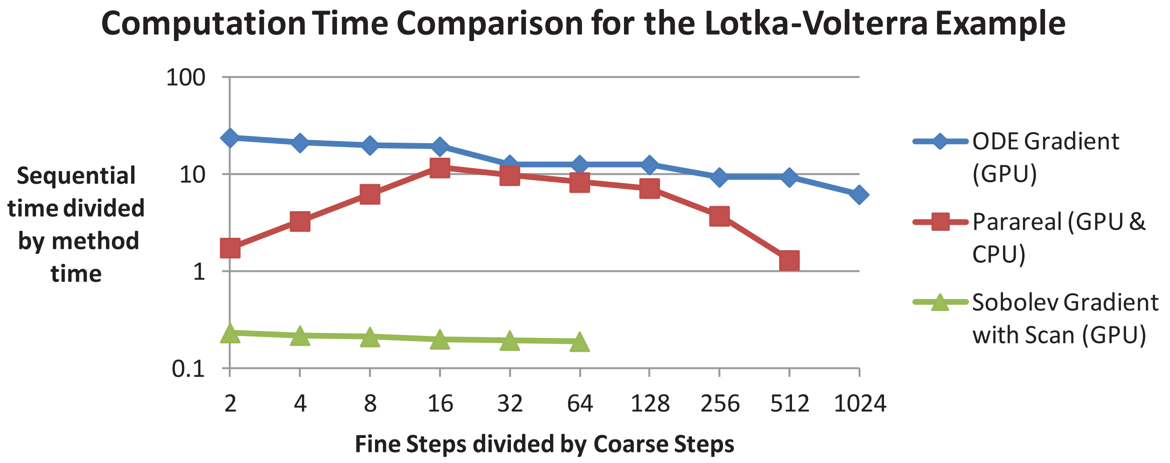

Figure 5. The computation time for each of the non-serial methods compared to the serial approach. The fine time step number is fixed at 32768 while the coarse step number is varied. The iterations are stopped when the error at the last time point is less than one tenth the error with 16384 time steps. Maximum speed up on system for completely parallel computations is 45.

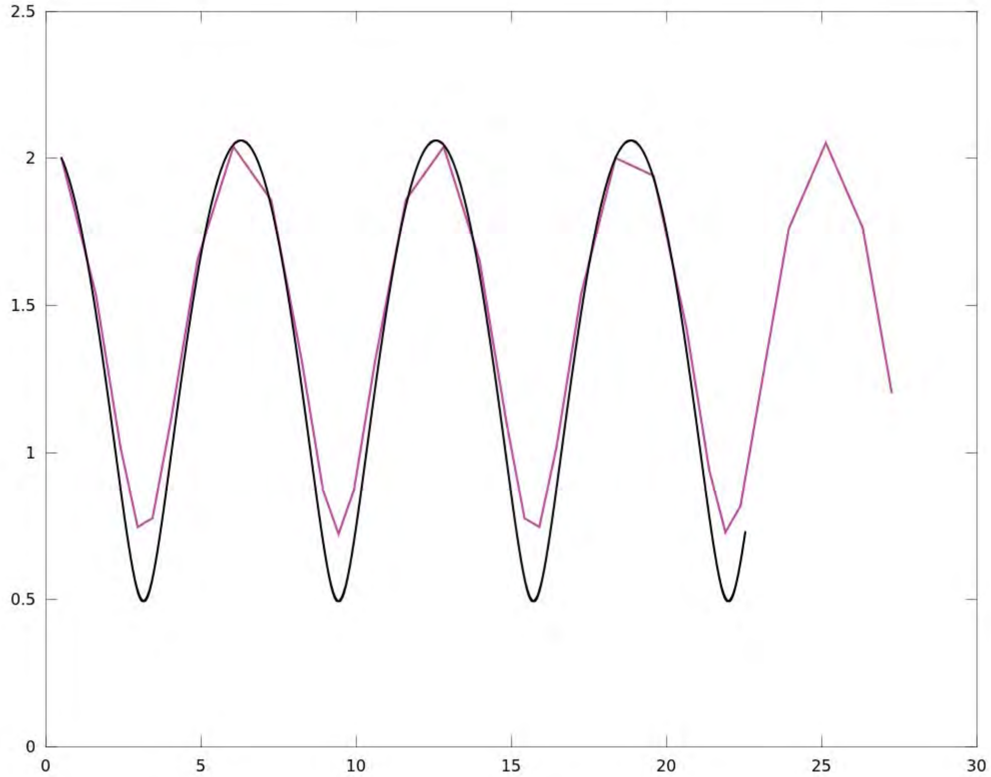

Figure 2. A phase space plot of the Pendulum example. Fine solution (black) contains 32768 time steps while the coarse solution (magenta) contains 32 time steps. The non-serial methods that converged are all visually indistinguishable from the fine solution computed in serial.

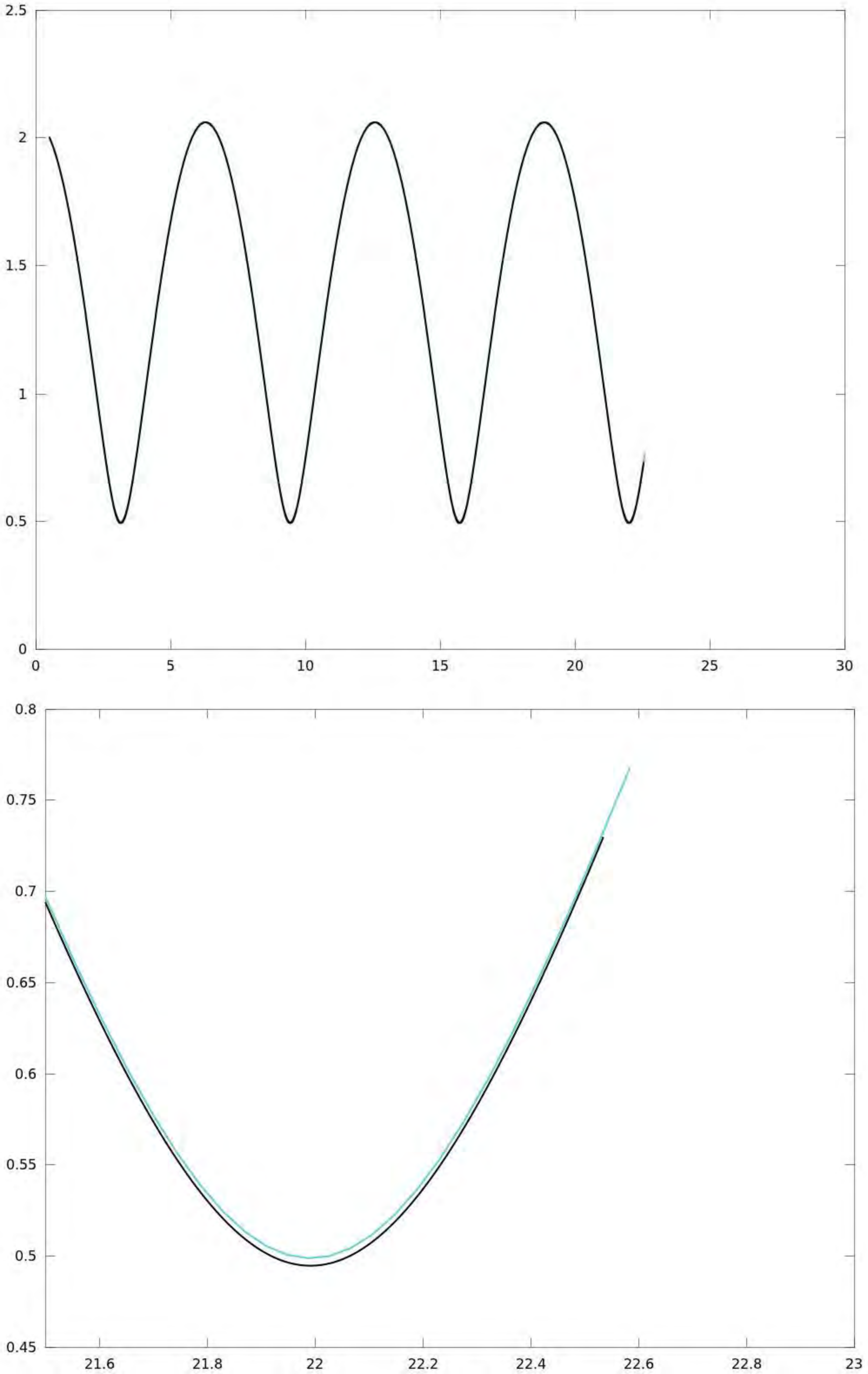

Figure 7. A phase space plot of the Pendulum example. Fine solution (black) contains 32768 time steps while the coarse solution (cyan) contains 256 time steps. The non-serial methods that converged are all visually indistinguishable from the fine solution computed in serial.

Distribution A: Approved for public release; distribution unlimited

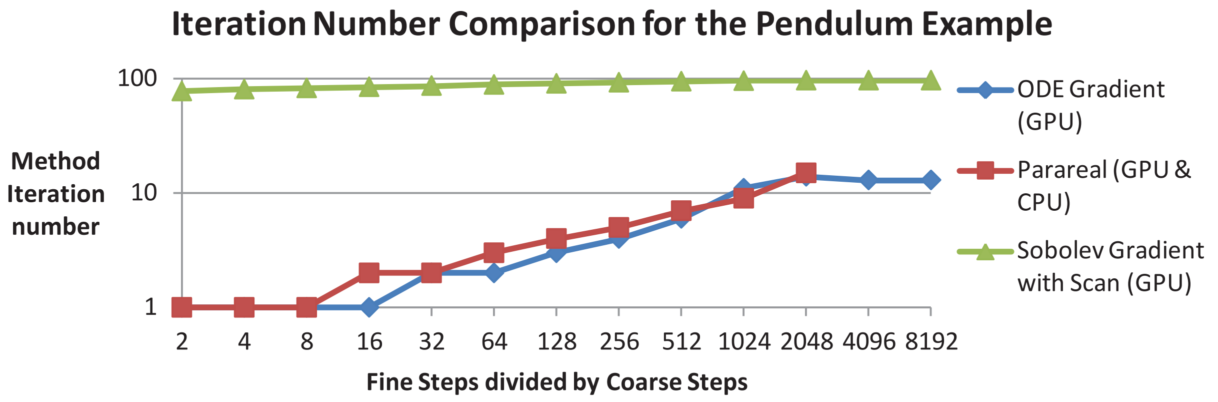

Figure 8. The number of iterations required for each of the non-serial methods. The fine time step number is fixed at 32768 while the coarse step number is varied. The iterations are stopped when the error at the last time point is less than one tenth the error with 16384 time steps.

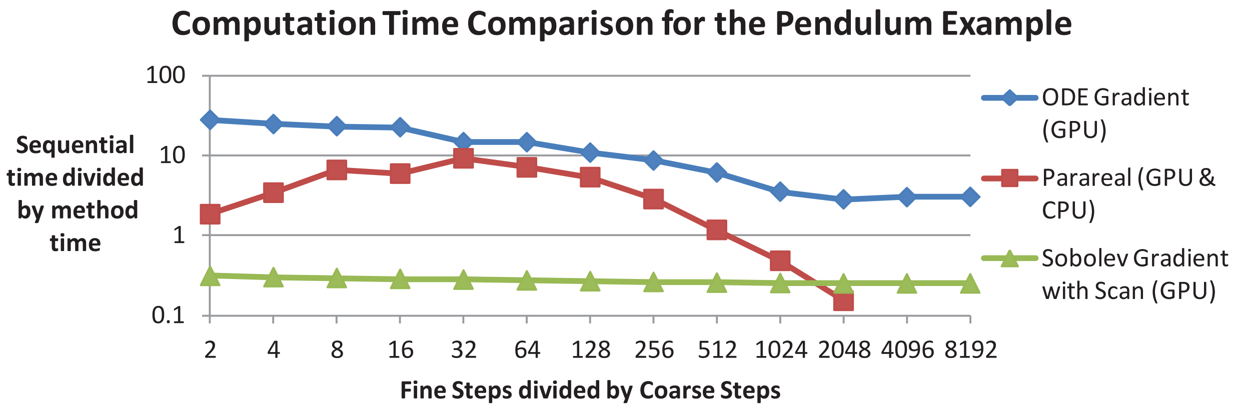

Figure 9. The computation time for each of the non-serial methods compared to the serial approach. The fine time step number is fixed at 32768 while the coarse step number is varied. The iterations are stopped when the error at the last time point is less than one tenth the error with 16384 time steps. Maximum speed up on system for completely parallel computations is 45.

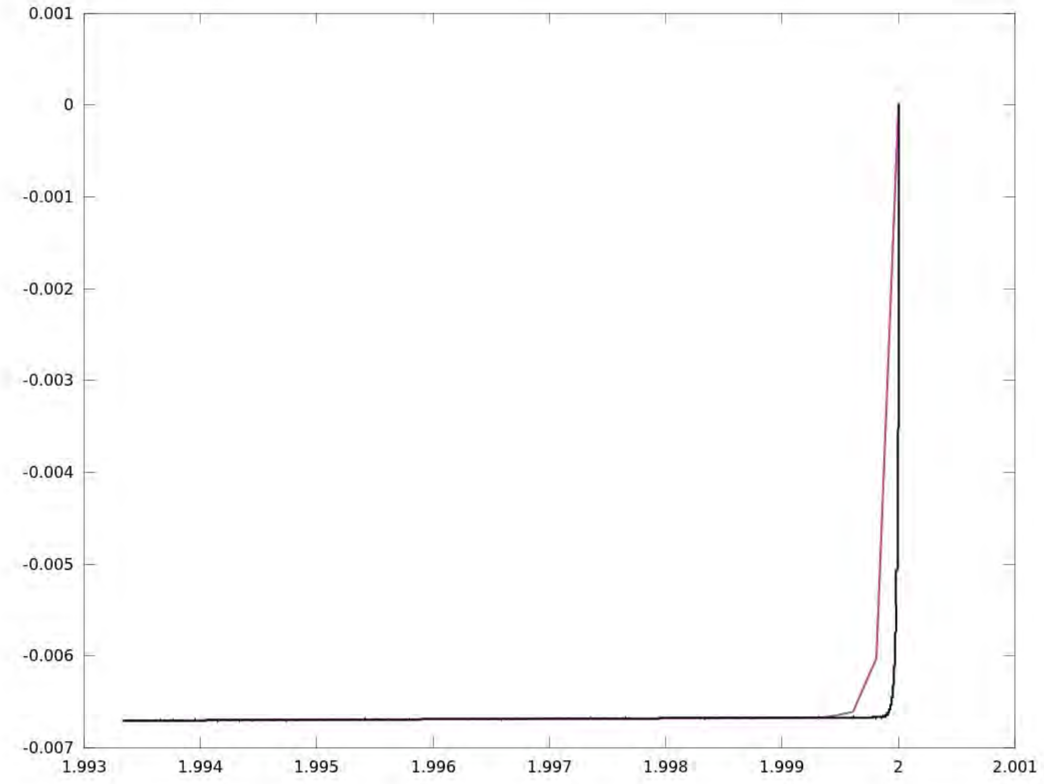

Figure 3. A phase space plot of the Van der Pol example. Fine solution (black) contains 32768 time steps while the coarse solution (magenta) contains 32 time steps. The non-serial methods that converged are all visually indistinguishable from the fine solution computed in serial.

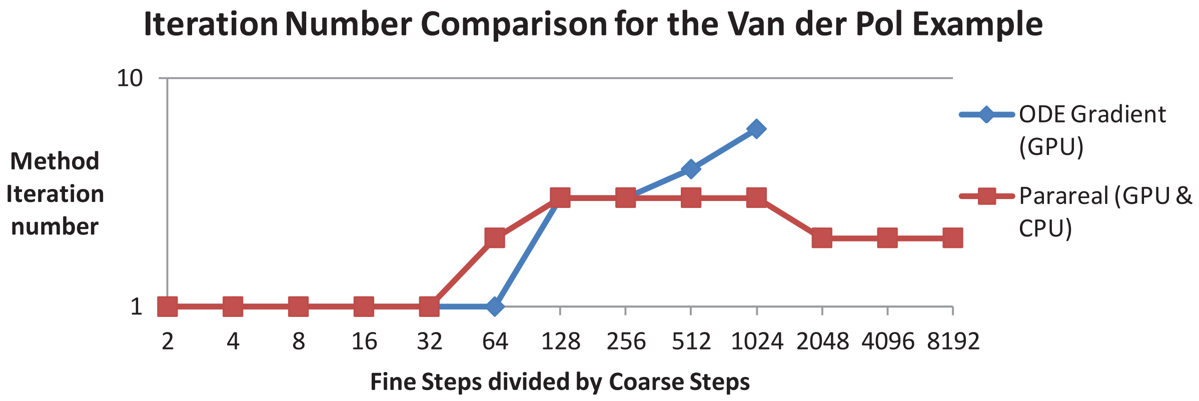

Figure 11. The number of iterations required for the Parareal and Proposed method. The fine time step number is fixed at 32768 while the coarse step number is varied. The iterations are stopped when the error at the last time point is less than one tenth the error with 16384 time steps.

Distribution A: Approved for public release; distribution unlimited

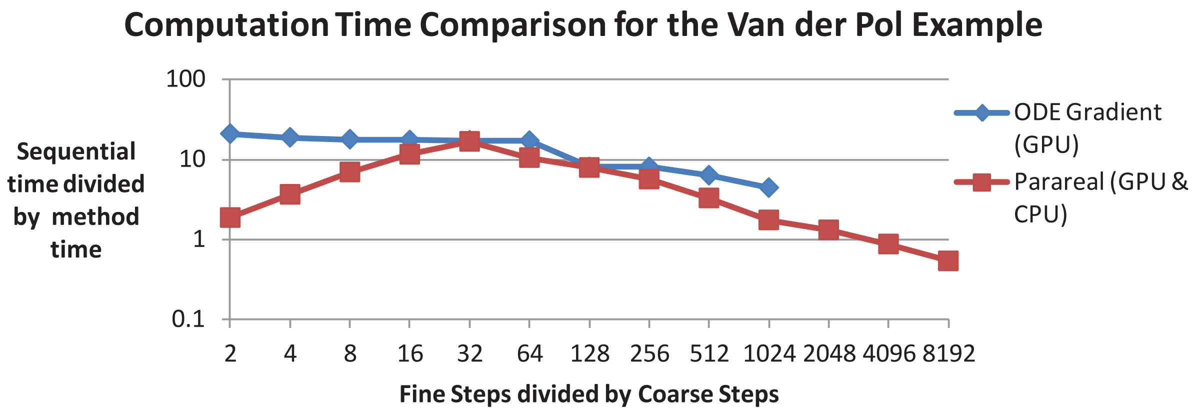

Figure 12. The computation time for the Parareal and Proposed method. The fine time step number is fixed at 32768 while the coarse step number is varied. The iterations are stopped when the error at the last time point is less than one tenth the error with 16384 time steps. Maximum speed up on system for completely parallel computations is 45.